



Apple. AppleTalk. Data Stream Protocol for VMS

Client Interface Specification
Release Note



• APPLE COMPUTER, INC.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or in part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© Apple Computer, Inc., 1988
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Apple, the Apple logo, AppleTalk, LaserWriter, and Macintosh are registered trademarks of Apple Computer, Inc.

Digital, DECnet, VAX, VMS, BLISS, and DIBOL are trademarks of Digital Equipment Corporation.

ITC Avant Garde Gothic, ITC Garamond, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated.

Varityper is a registered trademark, and VT600 is a trademark, of AM International, Inc.

Simultaneously published in the United States and Canada.

2/21/88



Contents

Chapter 1 Implementing ADSP 5

- ADSP functionality 6
- The VMS implementation 6
- Opening and closing a connection 7
- Sending and receiving data bytes 8
 - Breaking the data stream into logical messages 9
- Sending and receiving attention messages 9
- Forward-resetting the connection 10
- Implementing connection listeners 10
 - Establishing connection listeners 11
 - Receiving, accepting, and denying connection requests 11
- Obtaining connection status 12
- Tuning ADSP 12
- Breaking a connection 13
- Observing ADSP's internal significant events 14
- Synchronizing client operations with ADSP 14
- Buffer management 16
 - Buffer management internals 16
- Using tagged lists 17

Chapter 2 ADSP Call Formats 19

- ATK\$DSP_OPEN - Actively open a connection 21
- ATK\$DSP_LISTEN - Passively open a connection 26
- ATK\$DSP_CREATE_CON_END - Create a connection end 31
- ATK\$DSP_ESTABLISH - Establish a connection with a remote end 33
- ATK\$DSP_CLOSE - Close a connection 37
- ATK\$DSP_READ - Read data from a connection 39
- ATK\$DSP_WRITE - Write data to a connection 42
- ATK\$DSP_PUT_BYTE - Send a single byte 44
- ATK\$DSP_GET_BYTE - Get a single byte 45
- ATK\$DSP_SEND_EOM - Send an end-of-message marker 46

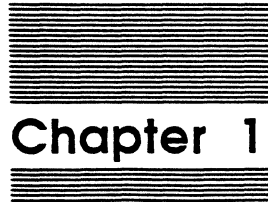
ATK\$DSP_FLUSH – Flush the send queue 47
 ATK\$DSP_SEND_ATTENTION – Send an attention message 48
 ATK\$DSP_RECV_ATTENTION – Receive an attention message 51
 ATK\$DSP_FORWARD_RESET – Forward-reset the connection 54
 ATK\$DSP_CREATE_LIS_SOC – Create a connection-listening socket 56
 ATK\$DSP_DELETE_LIS_SOC – Delete a connection-listening socket 57
 ATK\$DSP_ACCEPT – Accept a connection with a remote end 58
 ATK\$DSP_GET_CON_REQ – Get a connection request 64
 ATK\$DSP_DENY_CON_REQ – Deny a connection request 67
 ATK\$DSP_GET_BUF – Get a transmission buffer 69
 ATK\$DSP_RET_BUF – Return a receive buffer 70
 ATK\$DSP_SET_DEFAULTS – Set ADSP default operating parameters 71
 ATK\$DSP_GET_DEFAULTS – Get ADSP default operating parameters 72
 ATK\$DSP_GET_STATUS – Get connection status 73
 ATK\$DSP_GET_COUNTERS – Get ADSP internal counters 74
 ATK\$DSP_CLEAR_COUNTERS – Clear the ADSP internal counters 75

Appendix A Tuning Parameter Identifiers A-77

ATK\$K_ATTNBCKOFF A-78
 ATK\$K_ATTNINTRVL A-78
 ATK\$K_FWDRSTBCKOFF A-78
 ATK\$K_FWDRSTINTRVL A-79
 ATK\$K_OPENINTRVL A-79
 ATK\$K_OPENMAXRETRY A-79
 ATK\$K_RCVQCPYTHRESH A-80
 ATK\$K_RCVQMAXSIZE A-80
 ATK\$K_RESTARTBCKOFF A-80
 ATK\$K_RESTARTINTRVL A-81
 ATK\$K_RETRANSBCKOFF A-81
 ATK\$K_RETRANSINTRVL A-82
 ATK\$K_RETRANSTHRESH A-82
 ATK\$K_SNDQBUFSIZE A-82
 ATK\$K_SNDWDWCLOBCKOFF A-83
 ATK\$K_SNDWDWCLOINTRVL A-83
 ATK\$K_SNDWDWCLOTHRESH A-84
 ATK\$K_TRANSTHRESH A-84
 ATK\$K_TRANSTIMERINTRVL A-85

Appendix B Connection Status Identifiers B-87

Appendix C ADSP Internal Counter Identifiers C-89



Chapter 1

Implementing ADSP

This release note is an addendum to the AppleTalk® for VMS™ documentation suite, and describes new capabilities of the AppleTalk for VMS software, version 2.0. Version 2.0 of AppleTalk for VMS replaces version 1.6, and differs from it only in that the protocol support library now includes the AppleTalk Data Stream Protocol (ADSP).

The software described in this note is the VMS™ client interface to ADSP. This note therefore provides details about ADSP implementation only when needed to explain the ADSP interface. For a more detailed specification of ADSP, consult the *AppleTalk Data Stream Protocol Preliminary Note*, available from Apple Computer, Inc.

ADSP functionality

ADSP is a symmetric, connection-oriented protocol that makes possible the establishment and maintenance of full-duplex streams of data bytes between two sockets in an AppleTalk internet. ADSP ensures reliable data flow by delivering data bytes free of duplicates, in the same order as they are sent. In addition, ADSP includes a flow-control mechanism that uses information supplied by the intended destination socket.

Using ADSP, the client can open a connection between a local socket and a remote socket, send data to and receive data from the remote socket, and close the connection. The client can either send a continuous stream of data or logically break the data into client-intelligible messages. Additionally, ADSP provides an attention-message mechanism that the client can use for its own internal control. A forward-reset mechanism allows the client to abort the delivery of an outstanding stream of bytes to the remote client.

The ADSP protocol's connection-opening features include the ability to divert connection establishment to a third party, and the ability to limit the sockets with which a connection may be established. The client may also bypass the internal ADSP connection-opening mechanism by substituting an external, client-supplied mechanism.

The VMS implementation

The VMS implementation of ADSP is a set of procedure calls that provide access to all of the functionality of ADSP. Specifically, procedure calls are implemented that provide for

- ☐ opening and closing a connection
- ☐ sending/receiving data bytes
- ☐ breaking the data stream into logical messages

- ☐ sending/receiving attention messages
- ☐ forward-resetting the connection
- ☐ implementing connection listeners
- ☐ receiving/accepting/denying connection requests
- ☐ obtaining connection status

Several additional calls are provided for tuning ADSP, observing ADSP's internal significant events, and managing buffers.

Opening and closing a connection

Connection opening is the process of creating an association between two sockets in an internet to allow the reliable exchange of data streams. A socket and the connection-end information with which it is associated constitute a *connection end*. Connection-end information includes the connection identifier (CID) and the status of connection state variables.

Normally, the opening process requires a connection-establishment dialog between the two sockets to synchronize each connection end's state variables. State variables include the size of each end's receive queues and the initial sequence numbers of the data streams. The VMS implementation provides three procedure calls to establish a connection: `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, and `ATK$DSP_ESTABLISH`. Each procedure implements a different method of establishing a connection, based on the connection initiator and the availability of connection-end synchronizing information prior to establishing the connection.

`ATK$DSP_OPEN` is used to establish a connection with a specified remote socket. It actively seeks to establish a connection by initiating the opening dialog between the local and remote ADSP modules. No prior knowledge of any connection-end state variables is required to use this procedure.

`ATK$DSP_LISTEN` is the functional opposite of `ATK$DSP_OPEN`. Rather than actively seeking to open the connection, `ATK$DSP_LISTEN` creates a connection end on a socket and waits for some other remote socket to initiate the opening dialog.

`ATK$DSP_LISTEN` also does not require any prior knowledge of connection-end state variables.

Both `ATK$DSP_OPEN` and `ATK$DSP_LISTEN` allow the client to select the remote sockets with which the connection may be established. The connection initiator may specify a list of address filters against which a designated socket will be checked prior to connection establishment. Connection-denial messages are sent to those sockets which are not acceptable.

ATK\$DSP_ESTABLISH forces a connection to become immediately established. Within ADSP, no connection-opening dialog is initiated or received, and the client must provide all the information necessary to place the local connection end in a synchronized state. The client is also responsible for synchronizing the remote connection end. This form of connection establishment bypasses the ADSP connection-opening mechanism and allows the client to use a connection-opening dialog external to ADSP.

Prior to establishing a connection using ATK\$DSP_ESTABLISH, an ADSP client must first create a connection end upon which to establish a connection. The procedure call ATK\$DSP_CREATE_CON_END creates a connection end and returns the information necessary for the client to establish the connection at the remote end.

- ❖ *Note:* Even when a connection is explicitly denied by the remote ADSP module, the ATK\$DSP_OPEN procedure continues to try to establish the connection. The number of retries is specified by the opening parameter ATK\$K_OPENMAXRETRY.

When establishing a connection, a client may also specify the address of a procedure that will be called whenever a significant event occurs on the connection. Once the connection is actually established, this connection-event handler will be called under the following circumstances:

- ☐ the connection receives a connection-close advice from the remote end
- ☐ the connection is torn down because the remote end is unreachable
- ☐ a forward-reset message is received
- ☐ an attention message is received.

To break down a connection, the procedure ATK\$DSP_CLOSE is used. ATK\$DSP_CLOSE can be directed to break the connection immediately, or to wait until the send queues have emptied.

Sending and receiving data bytes

Once the connection is established, the flow of data can be started. Four procedure calls are implemented for reading and writing data to the connection:

ATK\$DSP_READ, ATK\$DSP_WRITE, ATK\$DSP_GET_BYTE, and ATK\$DSP_PUT_BYTE.

ATK\$DSP_READ and ATK\$DSP_WRITE are used to transmit and receive contiguous buffers of data bytes. ATK\$DSP_PUT_BYTE and ATK\$DSP_GET_BYTE read and write a single byte at a time. The Put/Get Byte interface is nonblocking, while the Read/Write interface allows the client to post a read request and be notified when a buffer of data has been received.

Each pair of calls implies a different buffer allocation scheme. With the buffer read and write routines, the client must allocate and deallocate buffers. With the byte routines, ADSP will allocate and deallocate buffers as necessary on the client's behalf. These two methods are further described in the discussion of buffer management internals later on in the section "Buffer Management."

Breaking the data stream into logical messages

Depending on the application, the ADSP client may need to mark message boundaries in the data stream. When writing data to a connection using `ATK$DSP_WRITE`, the client may pass a flag to ADSP indicating that the last byte in the buffer is the last byte of a client-intelligible message. ADSP will then inform the remote connection end about the end of message by inserting a logical end of message marker into the data stream immediately following the last byte of the message. The remote client will be notified of the end of message when it extracts the logical end of message marker from the connection.

When using `ATK$DSP_PUT_BYTE` to transmit data, a client cannot indicate an end-of-message. In this case the client must make a call to `ATK$DSP_SEND_EOM` to indicate that the last byte inserted into the connection was the last byte of a logical message. `ATK$DSP_SEND_EOM` also works with `ATK$DSP_WRITE`.

Sending and receiving attention messages

A client may need to send a message to the remote client outside of the normal data stream. The ADSP protocol provides for such out-of-band transmissions with its attention-message mechanism. Two procedure calls are implemented that allow the client to send and receive attention messages: `ATK$DSP_SEND_ATTENTION` and `ATK$DSP_RECV_ATTENTION`.

The connection-event handler will notify the client asynchronously when an attention message has been received. In this way, the client need only post an attention-message read when notified that one is available.

Forward-resetting the connection

During normal processing, the client may detect that the information just sent to the remote client is no longer useful. To respond to this situation, ADSP provides a mechanism for resetting the connection, purging the receiving connection end of any data bytes not yet delivered, and informing the remote client that the connection has been reset. This forward-reset mechanism is implemented for VMS with the procedure `ATK$DSP_FORWARD_RESET`.

Implementing connection listeners

A *connection listener* is a client of ADSP. This client uses a connection-listening socket on which to implement a listener function, which will divert incoming open connection requests to other sockets, as shown in Figure 1-1 and described below.

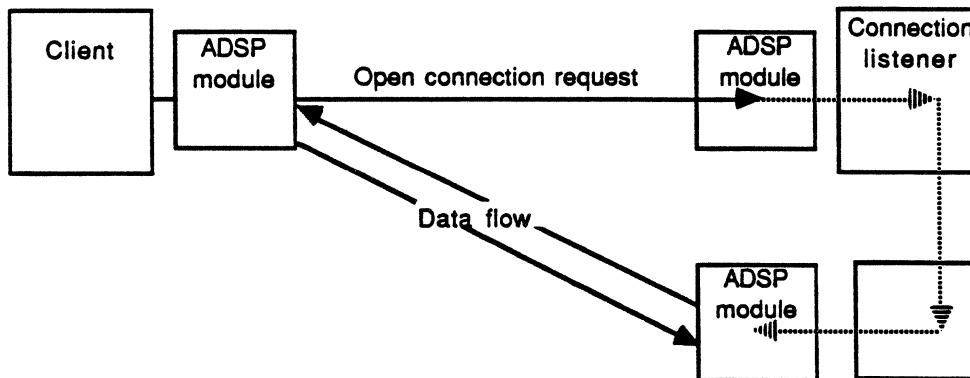


Figure 1-1
A connection listener implementation

A typical use of connection listeners is in the implementation of network-based servers (file, print, block, etc.). Such an implementation may require a separate copy of the server to be spawned for each active client-server session. A single process may field all open connection requests, clone a new copy of the requested server for each, and then divert each request to its new server. The server completes the connection with the remote client, and the data transfer may then begin. In this process, the connection listener uses the open connection request information to establish the actual connection on another socket, which may be on another node or network altogether.

Establishing connection listeners

The VMS implementation of ADSP provides two procedure calls for creating and deleting a connection-listening socket. These are `ATK$DSP_CREATE_LIS_SOC` and `ATK$DSP_DELETE_LIS_SOC`. Once a connection listener is established for a socket, future open connection requests received by that socket will be passed on intact to the client (normally, the ADSP implementation on the socket would handle open connection requests and responses internally, but the existence of the connection listener causes the request to pass to the client).

The ADSP protocol enables the connection listener to divert the open connection request to a different socket which can complete the connection with the requesting client.

When a connection listener is present, the initiator of an open connection request may be unable to ascertain in advance whether the target socket is the socket where the connection is finally established. Depending on the initiating client's application, this condition may not be acceptable. To solve this problem, the connection initiator may specify a list of address filters. When the open connection request is received by the connection initiator, the source socket is checked against the filter list. If the source socket is acceptable, the connection is established. Otherwise, a connection denial is transmitted.

Receiving, accepting, and denying connection requests

Three procedures are provided to connection listeners for receiving, accepting, and denying open connection requests. `ATK$DSP_GET_CON_REQ` is used to receive an open connection request from a listening socket. The information given to the client contains all the pertinent fields of the open connection request. `ATK$DSP_ACCEPT` is the procedure to use when the connection will be established on another socket. The client must supply the corresponding open connection request fields to this routine. `ATK$DSP_DENY_CON_REQ` will turn down an unacceptable connection request.

- ❖ **Note:** A connection request may be denied from either the connection listening socket or the newly created socket.

The VMS implementation does not restrict a connection listener from establishing a connection on the connection-listening socket. However, it does require that connection-listening operations be discontinued on the socket before the connection is established.

Obtaining connection status

The current status of a connection can be obtained by a call to the procedure `ATK$DSP_GET_STATUS`. The connection end's current state and sequence numbers are returned. A client may use this information to determine (among other things) whether the send queue has been emptied before closing a connection. The status information returned is described in Appendix B.

Tuning ADSP

Several implementation-specific operating parameters affect the performance of ADSP. Some of these parameters affect the local ADSP module's relationship with the remote ADSP module, while others affect the local ADSP module's internal management. Six procedure calls are provided to manipulate these parameters.

The first procedure, `ATK$DSP_GET_DEFAULTS`, is used to obtain a copy of the current default operating parameters. Two different lists may be obtained depending on the client's needs: the baseline list or the current operating defaults list.

The baseline list is the set of baseline operating parameters. This list can never be changed, and represents the factory-default settings for ADSP. A client can always refer to this list to get back to baseline operation. The client may also obtain a copy of this list from ADSP, modify selected fields, and use the new list to set up the current operating defaults list, or the operating parameters of a specific connection.

The second procedure, `ATK$DSP_SET_DEFAULTS`, is used to set the current operating defaults list. This list represents the operating parameters to be used for a connection if not explicitly supplied in the connection-establishment call. A client may set the operating parameters in this list, after which all future connections established will use this list.

The easiest way to use `ATK$DSP_SET_DEFAULTS` is to first obtain a full list from `ATK$DSP_GET_DEFAULTS`, modify the appropriate fields, and then set the current operating defaults.

The next four procedures are the actual connection-establishment procedures. `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, `ATK$DSP_ACCEPT`, and `ATK$DSP_ESTABLISH` all allow the client to specify the operating parameters that will be used to maintain the connection.

If the list is not specified, or parts of the list are missing, the current operating defaults list will be used. If the current operating defaults list is not specified, the baseline list is used. A separate copy of the operating parameters list exists for each active connection.

The routines which manipulate the tuning parameters use a special list to pass the necessary parameters back and forth. The format of the list and how it can be manipulated are described later in the section "Using Tagged Lists." Specific tuning parameters are discussed in Appendix A.

Breaking a connection

The breakdown of a connection begins with one of three events:

- ☐ The local client issues an `ATK$DSP_CLOSE` request.
- ☐ The local connection end receives a close advice message.
- ☐ The remote connection end becomes unreachable.

After any of these events, ADSP changes the state of the connection, reflecting the kind of event that began the breakdown. (The state of the connection is maintained internally by ADSP.) ADSP then attempts to notify the client that the event has occurred. This attempt succeeds only if the client has an active request posted to ADSP, or has an active connection-event handler. If this attempt to notify the client fails, the client is notified the next time it issues a request to ADSP.

During a breakdown, a client can perform only some of the operations that it normally can perform on a connection. Exactly which operations the client can perform depends on the breakdown mechanism—that is, on which kind of event begins the breakdown. If the breakdown begins because the client issued an `ATK$DSP_CLOSE` request whose wait parameter equals zero, the client can perform no operations on the connection. If, however, the wait parameter of the request does not equal zero, the client can perform only the operation `ATK$DSP_GET_STATUS`.

If the breakdown begins because the local connection end received a close advice message, or because the remote connection end became unreachable, the client can perform `ATK$DSP_GET_STATUS` and the following receive operations: `ATK$DSP_READ`, `ATK$DSP_GET_BYTE`, and `ATK$DSP_RECV_ATTENTION`.

The breakdown mechanism also governs the deletion of the send queues, receive queues, and local connection end. If the breakdown begins because the local client received a close advice message or because the remote connection end became unreachable, the send queues are deleted immediately. The receive queues, however, remain intact until the local client either (a) issues an `ATK$DSP_CLOSE` request, or (b) reads all data in the receive queue, and is notified of the breakdown.

If the breakdown begins because the local client issued an `ATK$DSP_CLOSE` request, the deletion of the connection end, send queues, and receive queues is governed by the wait parameter of the request. Unless the wait parameter equals zero, the send queues and receive queues are deleted only after the remote client acknowledges receipt of all sent data. The local connection end is then deleted after the queues are empty. If, on the other hand, the wait parameter equals zero, the send queues and receive queues are deleted immediately, including any data not yet delivered to the clients. The local connection end is deleted immediately after deletion of the queues.

Observing ADSP's internal significant events

ADSP for VMS provides two procedures for observing ADSP's internal significant events. `ATK$DSP_GET_COUNTERS` returns a list of ADSP counters. This list is a series of counters that describe the significant events that ADSP has incurred. `ATK$DSP_CLEAR_COUNTERS` is used to reset the counters. Appendix C describes each of the counters.

Synchronizing client operations with ADSP

The VMS implementation of ADSP allows the client to be notified of the completion of requested operations either synchronously or asynchronously. Some of the ADSP procedure calls, by their nature, can only be completed synchronously. These procedures return control to the client only after the requested operation is completed. The synchronous procedures are

- ☐ `ATK$DSP_CREATE_CON_END`
- ☐ `ATK$DSP_ESTABLISH`
- ☐ `ATK$DSP_WRITE`
- ☐ `ATK$DSP_PUT_BYTE`
- ☐ `ATK$DSP_GET_BYTE`
- ☐ `ATK$DSP_SEND_EOM`
- ☐ `ATK$DSP_FLUSH`
- ☐ `ATK$DSP_CREATE_LIS_SOC`
- ☐ `ATK$DSP_DELETE_LIS_SOC`
- ☐ `ATK$DSP_DENY_CON_REQUEST`
- ☐ `ATK$DSP_GET_BUF`
- ☐ `ATK$DSP_RET_BUF`
- ☐ `ATK$DSP_SET_DEFAULTS`

- ☐ ATK\$DSP_GET_DEFAULTS
- ☐ ATK\$DSP_GET_STATUS
- ☐ ATK\$DSP_GET_COUNTERS
- ☐ ATK\$DSP_CLEAR_COUNTERS

These procedures manipulate local data structures and, in some cases, may initiate an I/O operation. In all cases, control is returned quickly to the client. The client is informed of the success of the requested operation through the function call return mechanism. An example is ATK\$DSP_WRITE. A client making a call to ATK\$DSP_WRITE is requesting ADSP to place a buffer of data bytes in ADSP's internal send queue. Once the data is in the send queue, ADSP may or may not transmit the data immediately to the remote end, depending on the state of the remote end's receive queue. Regardless of whether the data is transmitted, control is returned to the client immediately and the operation is considered complete. The client can proceed with the assurance that the data will be received by the remote client at some point in the future.

Because the balance of the ADSP procedures could conceivably take a long time to complete, the client may wish to perform other functions while the procedure is in operation. Procedures that allow the client to work in parallel with the procedure's operations are

- ☐ ATK\$DSP_OPEN
- ☐ ATK\$DSP_LISTEN
- ☐ ATK\$DSP_ACCEPT
- ☐ ATK\$DSP_CLOSE
- ☐ ATK\$DSP_READ
- ☐ ATK\$DSP_SEND_ATTENTION
- ☐ ATK\$DSP_RECV_ATTENTION
- ☐ ATK\$DSP_FORWARD_RESET
- ☐ ATK\$DSP_GET_CON_REQ

The client may choose to wait for these procedures to complete before continuing, or to operate in parallel with the procedure's operation. If the client chooses to operate in parallel, a mechanism is provided that notifies the client when the operation has completed, and whether it has succeeded.

Four parameters can be provided to each of the procedures described above, which serve to synchronize the client's operation with the procedure:

- ☐ CompStatus is the address of a longword that will receive the completion status of a requested operation. Since asynchronous ADSP procedures return control to the client immediately after initiation, the CompStatus parameter is used to acquire their final completion status.

- **EventFlag** is the number of a VMS event flag. The specified event flag is cleared when the operation is initiated, and set when the operation is completed.
- **ASTHandler** is the address of a client procedure that will be called once the operation is complete. This procedure is executed at AST level and is passed the parameter **ASTParam** by value. The client may use **ASTHandler** to perform any client-specific cleanup.
- **ASTParam** is any client-specified value that will fit in a single longword. The value is passed to **ASTHandler** once the operation completes. This parameter may be used by the client with a general purpose **ASTHandler** procedure to distinguish which among many active operations has actually completed.

To use ADSP procedures in parallel, a client must specify either an **EventFlag**, an **ASTHandler**, or both. If neither are specified, the procedures will run to completion before returning control to the client.

Buffer management

When using **ATK\$DSP_READ** and **ATK\$DSP_WRITE**, the client is required to allocate buffers for transmission and to deallocate buffers that have been delivered to the client.

Two procedures are provided for client buffer management. **ATK\$DSP_GET_BUF** is used to allocate a buffer from the memory pool, which the client uses for data transmission. **ATK\$DSP_RET_BUF** is used to deallocate buffers obtained from **ATK\$DSP_GET_BUF**, **ATK\$DSP_READ**, or **ATK\$DSP_RECV_ATTENTION**.

Buffer management internals

The VMS implementation of ADSP provides two interfaces for transmitting to, and receiving data from a remote client in the normal data stream. The Put/Get Byte interface allows the transmission/reception of a single byte of data. The Read/Write interface allows buffers of multiple data bytes to be transmitted and received. The Put/Get Byte interface is nonblocking, while the Read/Write interface allows the client to post a read request and be notified when a buffer of data has been received.

When using the Put/Get Byte interface, all buffer management is performed by ADSP on the client's behalf. In actual implementation, ADSP maintains a linked list of buffers as the send/receive queues. The first time a client issues a Put Byte request, ADSP allocates a buffer from a memory pool which it shares with the client, and inserts the buffer at the end of the send queue linked list. The size of the buffer is specified by the client as a tuning parameter. ADSP then copies the data byte to the buffer; succeeding Put Byte requests will continue to fill up the original buffer. Once the buffer is filled, a new one is allocated and placed at the end of the link list.

As bytes are removed from the send queue (by remote client acknowledgements), ADSP checks for empty buffers. Emptied buffers are returned to the memory pool.

When bytes are received from the remote client in DDP (Datagram Delivery Protocol) data packets, the packets are linked into the tail of the receive queue. Each time the client issues a Get Byte request, the next byte at the head of the queue is delivered. As data packets are emptied, they are returned to DDP. A Get Byte request to an empty queue returns an error code.

Since a DDP data packet has a capacity of 586 bytes, and the entire packet is linked into the receive queue, there is a potential for wasted space if the remote client delivers only a small number of bytes in each packet. ADSP allows the client to specify the maximum acceptable level of waste via a tuning parameter. This parameter sets the byte count level at which ADSP will copy data bytes from the next received packet into the previous packet, rather than just linking the new packet into the receive queue. The parameter is specified as a byte count threshold. Packets with fewer bytes than this threshold are copied, while packets exceeding the threshold are linked.

When using the Read/Write interface, the buffer management strategy perceived by the client is that of a common memory pool. A client wishing to transmit data first allocates a buffer from the memory pool, fills it with data, then requests ADSP to transmit the data. At this point, ADSP accepts the buffer into its send queue for transmission. The buffer is now part of the send queue and the client may no longer access it. Eventually, the data will be received by the remote client and ADSP will release the buffer back to the memory pool.

When the client issues a Read request, the next DDP packet at the head of the receive queue is unlinked and delivered to the client, along with a pointer to the start of the data in the packet. When finished with the packet, the client requests ADSP to return the buffer to the memory pool. If the receive queue is empty when the client issues a read request, the request is posted; it completes when ADSP receives the next DDP packet from the remote end.

Using tagged lists

Tagged lists are used to pass a large number of parameters of different lengths and meanings through a procedure call interface. A *tagged list* is a sequence of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the entry field. The length of each list is defined as a constant.

ADSP uses tagged lists to pass default operating parameters, connection status, and ADSP internal counters back and forth to the client.

The number of parameters passed through a procedure call via a tagged list may be increased without affecting the caller of the procedure. From the caller's point of view, only the size of the list will have grown. If the caller has no use for the new parameters at the end of the list, they may simply be ignored. Provided the caller does not make any assumptions about the order of the parameters in the list—but uses tags for identification—the order of parameters can be changed by ADSP without affecting the client.



Chapter 2



ADSP Call Formats

This chapter provides call formats for the routines that follow.

ATK\$DSP_ACCEPT
ATK\$DSP_CLEAR_COUNTERS
ATK\$DSP_CLOSE
ATK\$DSP_CREATE_CON_END
ATK\$DSP_CREATE_LIS_SOC
ATK\$DSP_DELETE_LIS_SOC
ATK\$DSP_DENY_CON_REQ
ATK\$DSP_ESTABLISH
ATK\$DSP_FLUSH
ATK\$DSP_FORWARD_RESET
ATK\$DSP_GET_BUF
ATK\$DSP_GET_BYTE
ATK\$DSP_GET_CON_REQ
ATK\$DSP_GET_COUNTERS
ATK\$DSP_GET_DEFAULTS
ATK\$DSP_GET_STATUS
ATK\$DSP_LISTEN
ATK\$DSP_OPEN
ATK\$DSP_PUT_BYTE
ATK\$DSP_READ
ATK\$DSP_RECV_ATTENTION
ATK\$DSP_RET_BUF
ATK\$DSP_SEND_ATTENTION
ATK\$DSP_SEND_EOM
ATK\$DSP_SET_DEFAULTS
ATK\$DSP_WRITE

In addition to the call formats, this chapter provides a brief description of each routine and explains each of its arguments.

ATK\$DSP_OPEN – Actively open a connection

ATK\$DSP_OPEN actively seeks to open a connection with the specified socket.

Format	ATK\$DSP_OPEN	Refnum,RemoteAddress,CompStatus [,Socket] [,FilterList] [,ConEventHandler] [,EventFlag] [,ASTHandler] [,ASTParam] [,OPDefaults]
---------------	---------------	---

Arguments	Refnum
-----------	--------

```
type:      longword
access:    write only
mechanism: by reference
```

The connection reference number. Refnum identifies the connection end for all future operations on the connection. Refnum returns its value immediately upon the submission of the open call, enabling the client to use this information prior to the completion of the routine. If the open call is made synchronously, it then completes synchronously and returns all completion values at once.

RemoteAddress

```

type:      unspecified
access:    modify
mechanism: by reference

```

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

CompStatus

```

type:      longword
access:    write only
mechanism: by reference

```

A longword that receives the results of attempting to establish the connection. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the address of Socket is specified as zero or is not supplied, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is zero, ADSP will open a dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. If the address of the Socket argument is supplied, and is nonzero, the connection socket number will be returned. The Socket argument returns its value immediately upon the submission of the open call, enabling the client to use this information prior to the completion of the routine. If the open call is made synchronously, it then completes synchronously and returns all completion values at once.

FilterList

type: unspecified
access: read only
mechanism: by reference

The list of acceptable internet addresses with which ADSP may establish a connection. Each element of the list is a 4-byte quantity; the first two bytes contain an acceptable network address, the third byte contains an acceptable node number, and the fourth byte an acceptable socket number. A zero may be used as a wildcard character in either the network, node, or socket fields. The effect of a wildcard is to match any number in the field. Therefore, an entry with the network, node, and socket fields all equal to zero will match any internet address. An entry with the network field equal to three, the node field equal to five, and the socket field set to zero will match any socket on node number five in network number three. The list is scanned from the top down (which will affect which internet addresses are accepted) and is terminated with an entry with the network, node, and socket fields set to -1. If FilterList is omitted, ADSP will establish a connection with any internet address.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format	ConEventHandler	SigEvent, Refnum
---------------	-----------------	------------------

Arguments	SigEvent
------------------	----------

type: longword
access: read only
mechanism: by value

The type of significant event that has occurred. SigEvent will have one of four values: ATK__ATTMSGRCV, which indicates the receipt of an attention message; ATK__FORRESRCV, which indicates the receipt of a forward reset; ATK__CLOADVRCV, which indicates the receipt of a close advice from the remote end; and ATK__REMENDUNR, which indicates the connection being torn down due to inability to reach the remote end.

Refnum

type: longword
access: read only
mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

ConEventHandler is called as an AST routine. If ConEventHandler is omitted, then no connection event handler is called when a significant event occurs.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection is established. ASTHandler is called as normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection is established. ASTParam is passed by value to ASTHandler and may be used to identify the connection which has just become established (or for any other client-specified purpose). If ASTParam is not specified, the value zero is passed to ASTHandler.

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to `ATKSDSP_SET_DEFAULTS`). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__CONTABFUL	The connection table is full.
	ATK__CONLISACT	A connection listener is active on the specified socket.
	ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
	ATK__BADTUNPAR	Bad tuning parameter.
	ATK__BADPARLEN	Bad parameter length.
	ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYSSCLREF

LIB\$GET_VM

ATK\$DDP_OPN_SKT

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__CONOPNFAI	The connection opening failed.
ATK__CONDENIED	The connection was denied by the remote connection end.
ATK__UNAREMADR	Unacceptable address for the remote connection end.

ATK\$DSP_LISTEN – Passively open a connection

ATKSDSP_LISTEN creates a connection end, then waits for an incoming connect request from a remote socket.

Format	ATK\$DSP_LISTEN	Refnum,Socket,CompStatus [,RemoteAddress]
		[,FilterList] [,ConEventHandler] [,EventFlag]
		[,ASTHandler] [,ASTParam] [,OPDefaults]

Arguments	Refnum
-----------	--------

```

type:      longword
access:    write only
mechanism: by reference

```

The connection reference number. Refnum identifies the connection end for all future operations on the connection. The value of Refnum is available immediately upon return from this call, regardless of whether the call was issued synchronously.

Socket

```

type:      byte
access:    modify
mechanism: by reference

```

The DDP socket number through which the connection is to be established. If the value of Socket is zero, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. The connection socket number will be returned. If the Socket argument is omitted, an error is returned. The value of Socket is available immediately upon return from this call, regardless of whether the call was issued synchronously.

CompStatus

```

type:      longword
access:    write only
mechanism: by reference

```

A longword that receives the results of attempting to establish the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

RemoteAddress

type: unspecified
access: write only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

FilterList

type: unspecified
access: read only
mechanism: by reference

The list of acceptable internet addresses with which ADSP may establish a connection. Each element of the list is a 4-byte quantity; the first two bytes contain an acceptable network address, the third byte contains an acceptable node number, and the fourth byte an acceptable socket number. A zero may be used as a wildcard character in either the network, node, or socket fields. The effect of a wildcard is to match any number in the field. Therefore, an entry with the network, node, and socket fields all equal to zero will match any internet address. An entry with the network field equal to three, the node field equal to five, and the socket field set to zero will match any socket on node number five in network number three. The list is scanned from the top down (which will affect which internet addresses are accepted) and is terminated with an entry with the network, node, and socket fields set to -1. If FilterList is omitted, ADSP will establish a connection with any internet address.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format ConEventHandler SigEvent, Refnum

Arguments **SigEvent**

type: longword
access: read only
mechanism: by value

The type of significant event that has occurred. SigEvent will have one of four values: ATK__ATTMSGRCV, which indicates the receipt of an attention message; ATK__FORRESRCV, which indicates the receipt of a forward reset; ATK__CLOADVRCV, which indicates the receipt of a close advice from the remote end; and ATK__REMENDUNR, which indicates the connection being torn down due to inability to reach the remote end.

Refnum

type: longword
access: read only
mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

ConEventHandler is called as an AST routine. If ConEventHandler is omitted, then no connection event handler is called when a significant event occurs.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection is established. ASTHandler is called as normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection is established. ASTParam is passed by value to ASTHandler and may be used to identify the connection which has just become established (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATKSDSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__CONTABFUL	The connection table is full.
	ATK__CONLISACT	A connection listener is active on the specified socket.
	ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
	ATK__BADTUNPAR	Bad tuning parameter.
	ATK__BADPARLEN	Bad parameter length.
	ATK__INVSKTNUM	Invalid socket number.
	Any conditions returned by	
	SYS\$CLREF	
	LIB\$GET_VM	
	ATK\$DDP_OPN_SKT	
	Completion status values returned:	
	SS\$_NORMAL	Normal successful completion.
	ATK__CONCLOLOC	The connection was closed by the local connection end.
	ATK__CONCLOREM	The connection was closed by the remote connection end.
	ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_CREATE_CON_END - Create a connection end

ATK\$DSP_CREATE_CON_END creates a connection end that will later be used to establish a connection (via ATK\$DSP_ESTABLISH).

Format ATK\$DSP_CREATE_CON_END Refnum, Socket, LocalCID

Arguments **Refnum**

type: longword
access: write only
mechanism: by reference

The connection reference number. Refnum identifies the connection end for all future operations on the connection.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the value of Socket is zero, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. The connection socket number will be returned. If the Socket argument is omitted, an error is returned.

LocalCID

type: word
access: write only
mechanism: by reference

The 16-bit identifier to be used for the local connection end. LocalCID uniquely distinguishes the connection end from other connection ends that may be attached to the same socket.

Conditions	SS\$ _NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__CONTABFUL	The connection table is full.
	ATK__CONLISACT	A connection listener is active on the specified socket.
	ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
	ATK__INVSKTNUM	Invalid socket number.
	Any conditions returned by	
	SYSSCLREF	
	LIB\$GET_VM	
	ATK\$DDP_OPN_SKT	

ATK\$DSP_ESTABLISH – Establish a connection with a remote end

ATKSDSP_ESTABLISH bypasses the internal ADSP connection-opening dialog and forces a connection end to the open state.

Format ATK\$DSP_ESTABLISH Refnum,RemoteAddress,RemoteCID,SendSeq,
RecvWdw,RecvSeq,AttnSendSeq,AttnRecvSeq,
[,ConEventHandler] [,OPDefaults]

Arguments	Refnum
-----------	--------

```
type:      longword
access:    read only
mechanism: by value
```

The connection reference number. Refnum must be the reference number of a connection end created by `ATKSDSP_CREATE_CON_END`; otherwise an error will be returned.

RemoteAddress

```

type:      unspecified
access:    read only
mechanism: by reference

```

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

RemoteCID

```

type:      word
access:    read only
mechanism: by value

```

The 16-bit identifier of the connection end that is the target of the Open request. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket.

SendSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next untransmitted data byte in the local connection end's send queue. If SendSeq is omitted, an error is returned.

RecvWdw

type: word
access: read only
mechanism: by value

The size of remote connection end's receive queue. If it is omitted, an error is returned.

RecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next data byte the local connection end will accept into its receive queue. If it is omitted, an error is returned.

AttnSendSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message which will be transmitted from the local connection end. If it is omitted, an error is returned.

AttnRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

ConEventHandler

type: procedure entry mask
 access: call without stack unwinding
 mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format	ConEventHandler	SigEvent, Refnum
---------------	-----------------	------------------

Arguments	SigEvent
------------------	----------

type: longword
 access: read only
 mechanism: by value

The type of significant event that has occurred. SigEvent will have one of four values: ATK__ATTMSGRCV, which indicates the receipt of an attention message; ATK__FORRESRCV, which indicates the receipt of a forward reset; ATK__CLOADVRCV, which indicates the receipt of a close advice from the remote end; and ATK__REMENDUNR, which indicates the connection being torn down due to inability to reach the remote end.

Refnum

type: longword
 access: read only
 mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

ConEventHandler is called as an AST routine. If ConEventHandler is omitted, then no connection event handler is called when a significant event occurs.

OPDefaults

type: unspecified
 access: read only
 mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATKSDSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__BADREFNUM	Bad connection reference number.
	ATK__INVCONSTA	Invalid connection state.
	ATK__BADTUNPAR	Bad tuning parameter.
	ATK__BADPARLEN	Bad parameter length.

ATK\$DSP_CLOSE – Close a connection

ATK\$DSP_CLOSE terminates all connection I/O and frees all resources allocated by a connection end.

[illegible]

Arguments	Refnum
-----------	--------

```
type:      longword
access:    read only
mechanism: by value
```

The reference number of the connection end to be closed. Refnum must refer to a connection end previously established with `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, `ATK$DSP_ACCEPT`, `ATK$DSP_ESTABLISH`, or `ATK$DSP_CREATE_CON_END`. If Refnum is not specified, or is specified as zero, an error is returned.

Wait

```
type:      longword
access:    read only
mechanism: by value
```

Specifies whether the connection is to be closed after all data bytes have been successfully received by the remote connection end, or closed immediately. A zero value for Wait forces ADSP to close the connection immediately, without waiting for the send or send attention message queues to empty. Setting Wait to a nonzero value will delay closing the connection until the send queues are empty. If Wait is not specified, ADSP will not wait for the send queues to empty.

EventFlag

```

type:      longword
access:    read only
mechanism: by value

```

The event flag number to set when the connection actually closes. The event flag is cleared immediately and set once the connection goes to the closed state. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
 access: call without stack unwinding
 mechanism: by value

The address of a routine to be called when the connection actually closes.
 ASTHandler is called as a normal AST routine and is passed the parameter
 ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
 access: read only
 mechanism: by value

The parameter to be passed to ASTHandler when the connection actually closes.
 ASTParam is passed by value to ASTHandler and may be used to identify the
 connection end which has just closed (or for any other client-specified purpose). If
 ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

SYSS\$CLREF

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_READ – Read data from a connection

ATK\$DSP_READ gets a buffer of data bytes received from the remote connection end.

Format	ATK\$DSP_READ	Refnum, Bufptr, BufLength, Dataptr, ByteCount, CompStatus [,EOM] [,EventFlag] [,ASTHandler] [,ASTParam]
---------------	---------------	---

Arguments	Refnum
-----------	--------

type:	longword
access:	read only
mechanism:	by value

The reference number of the connection end from which to read data. Refnum must refer to a connection end previously established with `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, `ATK$DSP_ACCEPT`, or `ATK$DSP_ESTABLISH`. If Refnum is omitted, an error is returned.

Bu f p t r

```

type:      longword
access:    write only
mechanism: by reference

```

A longword which will receive a pointer to the next buffer of data obtained from the connection. If it is omitted, an error is returned.

BuLength

```

type:      longword
access:    write only
mechanism: by reference

```

The length in bytes of the buffer containing the data. If it is omitted, an error is returned.

Dataptr

```

type:      longword
access:    write only
mechanism: by reference

```

A longword which will receive a pointer to where the data actually starts inside the buffer. If it is omitted, an error is returned.

ByteCount

type: longword
access: write only
mechanism: by reference

The number of data bytes in the buffer, starting at `Dataptr`. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to read data from the connection. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes. If `CompStatus` is omitted, an error is returned.

EOM

type: longword
access: write only
mechanism: by reference

The ADSP logical end-of-message flag. Upon completion of the read, if bit zero is set in `EOM`, then the last byte in the buffer is the last byte of a logical (client-intelligible) message.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the read completes. The event flag is cleared immediately and set once the connection goes to the open state. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the read completes. `ASTHandler` is called as a normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is not specified, the value zero is passed instead.

ASTParam

type: longword
 access: read only
 mechanism: by value

The parameter to be passed to ASTHandler when the read completes. ASTParam is passed by value to ASTHandler and may be used to identify the read which has just completed (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__FWDRSTPNT	The forward reset point was reached.

Any conditions returned by

SYSSCLREF

LIB\$GET_VM

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__FWDRSTPNT	The forward reset point was reached.

ATK\$DSP_WRITE - Write data to a connection

ATK\$DSP_WRITE inserts a buffer of data bytes into the send queue for transmission to the remote end.

Format	ATK\$DSP_WRITE	Refnum,Buffer,BufLength [,ByteCount] [,EOM] [,Flush]
---------------	----------------	---

Arguments	Refnum
-----------	--------

```
type:      longword
access:    read only
mechanism: by value
```

The reference number of the connection end into which the data will be inserted. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Buffer

```

type:      unspecified
access:    read only
mechanism: by reference

```

The buffer that contains the data to be inserted into the data stream. It must be the address of the beginning of a buffer obtained through an `ATKSDSP_GET_BUF`, `ATKSDSP_READ`, or `ATKSDSP_RECV_ATTENTION` call. The specified buffer becomes the exclusive property of ADSP and cannot be reused by the client. If `Buffer` is omitted when `BufLength` is nonzero, an error is returned.

BufLength

```
type:      longword
access:    read only
mechanism: by value
```

The length of Buffer in bytes. This value must be the length returned by a call to `ATK$DSP_GET_BUF`, `ATK$DSP_READ`, or `ATK$DSP_RECV_ATTENTION`. If `BufLength` is omitted, an error is returned. Zero is an acceptable value for `BufLength`, indicating that a logical end-of-message (and no data) is being inserted into the data stream, or that the send queue is being flushed. If `BufLength` is specified as zero, Buffer is not accessed.

ByteCount

type: longword
 access: read only
 mechanism: by value

The number of bytes contained in the buffer to be inserted into the data stream. This value may be any number from zero to BufLength. If it is not specified, or is specified as zero, the value of BufLength is used instead.

EOM

type: longword
 access: read only
 mechanism: by value

The logical end-of-message flag. If nonzero, a logical EOM is inserted into the data stream right after the last byte in the buffer. If EOM is not specified, or is specified as zero, no logical end-of-message is inserted into the data stream.

Flush

type: longword
 access: read only
 mechanism: by value

If specified as nonzero, all remaining data bytes in the send queue are transmitted to the remote connection end. If Flush is not specified, or is specified as zero, data bytes remaining in the send queue are transmitted at the appropriate time.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__BADPARAM	Bad parameter.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

LIB\$GET_VM

ATK\$DSP_PUT_BYTE – Send a single byte

ATK\$DSP_PUT_BYTE writes a single data byte to the send queue for transmission to the remote end.

Format ATK\$DSP_PUT_BYTE Refnum,Byte

Arguments **Refnum**

type: longword
access: read only
mechanism: by value

The reference number of the connection end to accept the byte for transmission to the remote end. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Byte

type: byte
access: read only
mechanism: by value

The value of the data byte to be inserted into the data stream. If it is omitted, an error is returned.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

LIB\$GET_VM

ATK\$DSP_GET_BYTE - Get a single byte

ATK\$DSP_GET_BYTE reads a single data byte from the receive queue. If the receive queue is empty, an error is returned.

Format ATK\$DSP_GET_BYTE Refnum,Byte

Arguments **Refnum**

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which to obtain the byte. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Byte

type: byte
access: write only
mechanism: by reference

The value of the next data byte available from the receive queue.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__BADREFNUM	Bad connection reference number.
	ATK__CONNOTOPN	The connection is not in the open state.
	ATK__CONCLOLOC	The connection was closed by the local connection end.
	ATK__CONCLOREM	The connection was closed by the remote connection end.
	ATK__CONTORDOW	The connection was torn down.
	ATK__FWDRSTPNT	The forward reset point was reached.
	ATK__LOGENDMES	A logical end of message byte was encountered.
	ATK__RCVQUEEMP	The receive queue is empty.

ATK\$DSP_SEND_EOM – Send an end-of-message marker

ATK\$DSP_SEND_EOM places an end-of-message marker in the send queue.

Format ATK\$DSP_SEND_EOM Refnum

Arguments Refnum

type: longword
 access: read only
 mechanism: by value

The reference number of the connection end into which the EOM will be inserted. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

LIB\$GET_VM

ATK\$DSP_FLUSH – Flush the send queue

ATK\$DSP_FLUSH transmits all data bytes remaining in the send queue.

Format ATK\$DSP_FLUSH Refnum

Arguments Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end whose send queue is to be flushed. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Conditions	SS\$ _NORMAL	Normal successful completion.
	ATK __INSNUMARG	Insufficient number of arguments supplied.
	ATK __BADREFNUM	Bad connection reference number.
	ATK __CONNOTOPN	The connection is not in the open state.
	ATK __CONCLOLOC	The connection was closed by the local connection end.
	ATK __CONCLOREM	The connection was closed by the remote connection end.
	ATK __CONTORDOW	The connection was torn down.

ATK\$DSP_SEND_ATTENTION - Send an attention message

ATK\$DSP_SEND_ATTENTION sends an attention message to the remote end. Only one attention message can be sent at a time.

Format	ATK\$DSP_SEND_ATTENTION Refnum,Buffer,BufLength,AttnCode, CompStatus [,ByteCount] [,EventFlag] [,ASTHandler] [,ASTParam]
---------------	--

Arguments	Refnum
-----------	--------

```
type:      longword
access:    read only
mechanism: by value
```

The reference number of the connection end from which the attention message will be sent. Refnum must refer to a connection end previously established with `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, `ATK$DSP_ACCEPT`, or `ATK$DSP_ESTABLISH`. If Refnum is omitted, an error is returned.

Buffer

```

type:      unspecified
access:    read only
mechanism: by reference

```

The buffer that contains the attention message data. It must be the address of the beginning of a buffer obtained through an `ATKSDSP_GET_BUF`, `ATKSDSP_READ`, or `ATKSDSP_RECV_ATTENTION` call. The buffer becomes the exclusive property of ADSP and cannot be reused by the client. If `Buffer` is omitted, an error is returned. Note that to send an attention message containing no data, `BufLength` and `ByteCount` must be set to zero.

BufLength

```

type:      longword
access:    read only
mechanism: by value

```

The length of Buffer in bytes. This value must be the length returned by a call to ATKSDSP_GET_BUF, ATKSDSP_READ, or ATKSDSP_RECV_ATTENTION. If it is omitted, an error is returned.

AttnCode

type: word
access: read only
mechanism: by value

The attention code to be placed in the ADSP header. Note that bit 16 of AttnCode is reserved for use by Apple Computer, Inc.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to send the attention message. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

ByteCount

type: longword
access: read only
mechanism: by value

The number of attention message data bytes contained in buffer. The value of ByteCount is limited by the maximum attention message size (572 bytes) or BufLength, whichever is smaller. If ByteCount is omitted, the value of BufLength is used instead.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the attention message has been sent. The event flag is cleared immediately and set when the attention message has been sent. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the attention message has been sent. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
 access: read only
 mechanism: by value

The parameter to be passed to ASTHandler when the attention message has been sent. ASTParam is passed by value to ASTHandler and may be used to identify the message that has just been sent (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__BADPARAM	Bad parameter.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__OPEALRPRO	The operation is already in progress.
ATK__ILLATTCOD	Illegal attention message code.

Any conditions returned by

SY\$_CLREF

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_RECV_ATTENTION – Receive an attention message

ATK\$DSP_RECV_ATTENTION is used to receive an attention message from the remote connection end.

Format ATK\$DSP_RECV_ATTENTION Refnum, Bufptr, BufLength, Dataptr, ByteCount, AttnCode, CompStatus, [,EventFlag] [,ASTHandler] [,ASTParam]

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which the attention message will be received. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Bufptr

type: longword
access: write only
mechanism: by reference

A longword to receive a pointer to the buffer containing the next attention message delivered to the connection end. If it is omitted, an error is returned.

BufLength

type: longword
access: write only
mechanism: by reference

The length in bytes of the buffer containing the attention message. If it is omitted, an error is returned.

ByteCount

type: longword
access: write only
mechanism: by reference

The number of attention message data bytes contained in the buffer, starting at Dataptr. If it is omitted, an error is returned.

AttnCode

type: word
access: write only
mechanism: by reference

The attention code from the ADSP header.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to receive an attention message. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If it is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when an attention message has been received. The event flag is cleared immediately and set once the attention message has been received. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when an attention message has been received. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when an attention message has been received. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just received an attention message (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__BADREFNUM	Bad connection reference number.
	ATK__CONNOTOPN	The connection is not in the open state.
	ATK__CONCLOLOC	The connection was closed by the local connection end.
	ATK__CONCLOREM	The connection was closed by the remote connection end.
	ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

SYSSCLREF

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_FORWARD_RESET - Forward-reset the connection

ATK\$DSP_FORWARD_RESET performs a forward reset on the connection.

Format	ATKSDSP_FORWARD_RESET	Refnum, CompStatus [, EventFlag] [, ASTHandler] [, ASTParam]
---------------	-----------------------	---

Arguments	Refnum
-----------	--------

```
type:      longword
access:    read only
mechanism: by value
```

The reference number of the connection end that will be forward-reset. Refnum must refer to a connection end previously established with `ATKSDSP_OPEN`, `ATKSDSP_LISTEN`, `ATKSDSP_ACCEPT`, or `ATKSDSP_ESTABLISH`. If Refnum is omitted, an error is returned.

CompStatus

```

type:      longword
access:    write only
mechanism: by reference

```

A longword that receives the results of attempting to forward-reset the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

EventFlag

```

type:      longword
access:    read only
mechanism: by value

```

The event flag number to set when the connection has been forward-reset. The event flag is cleared immediately and set once the forward reset has been accepted by the remote ADSP. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
 access: call without stack unwinding
 mechanism: by reference

The address of a routine to be called when the connection has been forward-reset. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
 access: read only
 mechanism: by value

The parameter to be passed to ASTHandler when the connection has been forward-reset. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just been forward-reset (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__OPEALRPRO	The operation is already in progress.

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_CREATE_LIS_SOC - Create a connection-listening socket

ATK\$DSP_CREATE_LIS_SOC creates a socket that can be used by a connection listener.

Format ATK\$DSP_CREATE_LIS_SOC Socket

Arguments **Socket**

type: byte
access: modify
mechanism: by reference

The socket number to use to create a connection-listening socket. If specified as zero, ADSP will request DDP to create a socket for the connection listener and return the number of the created socket. If Socket is omitted, an error is returned.

Conditions SS\$ _NORMAL Normal successful completion.

ATK__INSNUMARG Insufficient number of arguments supplied.

Any conditions returned by

LIB\$GET_VM

ATK\$DDP_OPN_SKT

ATK\$DSP_DELETE_LIS_SOC - Delete a connection-listening socket

ATK\$DSP_DELETE_LIS_SOC terminates socket I/O operations and frees all resources associated with the socket.

Format ATK\$DSP_DELETE_LIS_SOC Socket

Arguments **Socket**

type: byte
access: read only
mechanism: by value

The socket number of the connection-listening socket that will be deleted. If the DDP socket was created by ADSP, then the socket will be deleted. If not, the client must delete the socket. In either case, all connection-listening operations are terminated on the socket. If Socket is omitted, an error is returned.

Conditions	SS\$_NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__INVSKTNUM	Invalid DDP socket number.
	ATK__NOTCONLIS	Not a connection listening socket.

ATK\$DSP_ACCEPT – Accept a connection with a remote end

ATK\$DSP_ACCEPT accepts a connection with a remote socket based on parameters received by a connection listener.

Format	ATK\$DSP_ACCEPT	Refnum, RemoteAddress, RemoteCID, PktNextRecvSeq , PktRecvWdw, PktAttnRecvSeq, CompStatus [, Socket] [, ConEventHandler] [, EventFlag] [, ASTHandler] [, ASTParam] [, OPDefaults]
---------------	-----------------	--

Arguments	Refnum
-----------	--------

```

type:      longword
access:    write only
mechanism: by reference

```

The connection reference number. Refnum identifies the connection end for all future operations on the connection. The value of Refnum is available immediately upon return from this call, regardless of whether the call was issued synchronously.

RemoteAddress

```

type:      unspecified
access:    read only
mechanism: by reference

```

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

RemoteCID

```
type:      word
access:    read only
mechanism: by value
```

The 16-bit identifier of the connection end that is the target of the Open request. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket.

PktNextRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next data byte the remote connection end will accept into its receive queue. If it is omitted, an error is returned.

PktRecvWdw

type: word
access: read only
mechanism: by value

The size of remote connection end's receive queue. If it is omitted, an error is returned.

PktAttnRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to establish the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

Socket

type: byte
 access: modify
 mechanism: by reference

The DDP socket number through which the connection is to be established. If the address of Socket is specified as zero or is not supplied, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is zero, ADSP will open a dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. If the address of the Socket argument is supplied, and is nonzero, the connection socket number will be returned.

ConEventHandler

type: procedure entry mask
 access: call without stack unwinding
 mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format ConEventHandler SigEvent, Refnum

Arguments SigEvent

type: longword
 access: read only
 mechanism: by value

The type of significant event that has occurred. SigEvent will have one of four values: ATK__ATTMSGRCV, which indicates the receipt of an attention message; ATK__FORRESRCV, which indicates the receipt of a forward reset; ATK__CLOADVRCV, which indicates the receipt of a close advice from the remote end; and ATK__REMENDUNR, which indicates the connection being torn down due to inability to reach the remote end.

Refnum

type: longword
access: read only
mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

ConEventHandler is called as an AST routine. If ConEventHandler is omitted, then no connection event handler is called when a significant event occurs.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection is established. ASTHandler is called as normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection is established. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just become established (or for any other compatible purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

OPDefaults

type: unspecified
 access: read only
 mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry that has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATK\$DSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__CONTABFUL	The connection table is full.
ATK__CONLISACT	A connection listener is active on the specified socket.
ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.
ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYSS\$CLREF

LIB\$GET_VM

ATK\$DDP_OPN_SKT

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK__CONOPNFAI	The connection opening failed.
ATK__CONDENIED	The connection was denied by the remote connection end
ATK__UNAREMADR	Unacceptable address for the remote connection end.

ATK\$DSP_GET_CON_REQ - Get a connection request

ATK\$DSP_GET_CON_REQ gets the next connection request delivered to the listening socket.

Format

```
ATK$DSP_GET_CON_REQ Socket, RemoteAddress, RemoteCID,  
                    PktNextRecvSeq, PktRecvWdw,  
                    PktAttnRecvSeq, CompStatus [, EventFlag]  
                    [, ASTHandler] [, ASTParam]
```

Arguments

Socket

type: byte
access: read only
mechanism: by value

The socket number of the connection listening socket. If it is omitted, an error is returned.

RemoteAddress

type: unspecified
access: write only
mechanism: by reference

The 4-byte AppleTalk internet address of the connection end that issued the open connection request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. If RemoteAddress is omitted, an error is returned.

RemoteCID

type: word
access: write only
mechanism: by value

The 16-bit identifier which uniquely identifies the connection end that issued the open connection request. If it is not specified, an error is returned.

PktNextRecvSeq

type: longword
access: write only
mechanism: by value

The sequence number of the next data byte the remote connection end will accept into its receive queue. If it is not specified, an error is returned.

PktRecvWdw

type: word
access: write only
mechanism: by reference

The size in bytes of the remote connection end's receive window. If it is omitted, an error is returned.

PktAttnRecvSeq

type: longword
access: write only
mechanism: by reference

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to get a connection request. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when a connection request is received. The event flag is cleared immediately and set once a connection request is received. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection request is received. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
 access: read only
 mechanism: by value

The parameter to be passed to ASTHandler when the connection request is received. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just received an open request (or for any other client-specified purpose). If ASTParam is not specified, the value zero is passed to ASTHandler.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__INVSKTNUM	Invalid DDP socket number.
ATK__NOTCONLIS	Not a connection listening socket.
ATK__OPEALRPRO	The operation is already in progress.

Any conditions returned by

SY\$_CLREF

Completion status values returned:

SS\$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK\$DSP_DENY_CON_REQ - Deny a connection request

ATK\$DSP_DENY_CON_REQ sends a connection denial message to the remote end.

Format

ATK\$DSP_DENY_CON_REQ Socket, RemoteCID, RemoteAddress

Arguments

Socket

type: byte
access: read only
mechanism: by value

The DDP socket number through which to send the connection denial message. Socket must identify a socket that is currently open, otherwise, an error is returned. If Socket is not specified, or is specified as zero, an error is returned.

RemoteCID

type: word
access: read only
mechanism: by value

The 16-bit identifier of the connection end that is the target of the connection denial message. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket. If RemoteCID is not specified, or is specified as zero, an error is returned.

RemoteAddress

type: unspecified
access: read only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the connection denial message. The first two bytes contain the 16 bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. If RemoteAddress is not specified, or is specified as zero, an error is returned.

Conditions	SS\$ _NORMAL	Normal successful completion.
	ATK __INSNUMARG	Insufficient number of arguments supplied.
	ATK __INVSKTNUM	Invalid DDP socket number.
	ATK __INVCONID	Invalid connection identifier.
Any conditions returned by		
LIB\$GET_VM		

ATK\$DSP_GET_BUF – Get a transmission buffer

ATK\$DSP_GET_BUF gets a buffer from the memory pool, to be used for data transmission across the connection.

Format ATK\$DSP_GET_BUF Bufptr, BufLength

Arguments **Bufptr**

type: longword
access: read only
mechanism: by value

A longword that receives the address of the allocated buffer. If the specified amount of memory cannot be allocated, Bufptr remains unchanged. If the Bufptr argument is omitted, an error is returned.

BufLength

type: longword
access: read only
mechanism: by value

The length in bytes of the buffer to allocate from the memory pool. If it is not specified, or is specified as zero, an error is returned.

Conditions \$\$\$_NORMAL Normal successful completion.

Any conditions returned by

LIB\$GET_VM

ATK\$DSP_RET_BUF – Return a receive buffer

ATK\$DSP_RET_BUF returns a receive buffer obtained from ADSP to the memory pool.

Format ATK\$DSP_RET_BUF Bufptr, BufLength

Arguments **Bufptr**

type: longword
access: read only
mechanism: by value

A pointer to the buffer being returned to the memory pool. If the Bufptr argument is omitted, an error is returned.

BufLength

type: longword
access: read only
mechanism: by value

The length in bytes of the buffer being returned to the memory pool. If the Buflength argument is omitted, an error is returned.

Conditions SS\$_NORMAL Normal successful completion.

Any conditions returned by

LIB\$FREE_VM

ATK\$DSP_SET_DEFAULTS - Set ADSP default operating parameters

ATK\$DSP_SET_DEFAULTS sets the default operating parameters for ADSP. (Each future connection uses these operating defaults unless its connection-establishment call explicitly changes the defaults for that specific connection.)

Format ATK\$DSP_SET_DEFAULTS OPDefaults

Arguments OPDefaults

type: unspecified
 access: read only
 mechanism: by reference

A tagged list of the operating parameters ADSP is to use as the current default operating parameters. Each connection established after calling ATK\$DSP_SET_DEFAULTS will use the new default operating parameters for the connection, unless the connection establishment call provides its own set of operating parameters. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry that has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, an error is returned. See Appendix A for a list of operating parameters.

Conditions	SS\$ _NORMAL	Normal successful completion.
	ATK__INSNUMARG	Insufficient number of arguments supplied.
	ATK__BADTUNPAR	Bad tuning parameter.
	ATK__BADPARLEN	Bad parameter length.

ATK\$DSP_GET_DEFAULTS - Get ADSP default operating parameters

ATK\$DSP_GET_DEFAULTS obtains the current or baseline ADSP default operating parameters.

Format ATK\$DSP_GET_DEFAULTS OPDefaults [,Baseline]

Arguments OPDefaults

type: unspecified
 access: write only
 mechanism: by reference

A tagged list of the operating parameters ADSP is currently using. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK\$K_DefLen bytes in length. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP operating parameter follows the length field in the entry. If OPDefaults is omitted, an error is returned. See Appendix A for a list of operating parameters.

Baseline

type: longword
 access: read only
 mechanism: by value

A flag indicating whether the baseline or current default operating parameters are required. If the Baseline argument is nonzero, the baseline operating parameters are returned. If the Baseline argument is zero, or is not specified, the current operating defaults are returned.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__PARLISINV	The requested parameter list is invalid.

ATK\$DSP_GET_STATUS - Get connection status

ATK\$DSP_GET_STATUS obtains the current status of the specified connection end.

Format ATK\$DSP_GET_STATUS Refnum, ConStatus

Arguments **Refnum**

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which the status will be obtained. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

ConStatus

type: unspecified
access: write only
mechanism: by reference

A tagged list of parameters describing the current state of the connection and the network management counters for the connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK\$K_StaLen bytes in length. The first byte of each entry is the tag, identifying the contents of the entry. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the entry follows the length field. If ConStatus is omitted, an error is returned.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.

ATK\$DSP_GET_COUNTERS - Get ADSP internal counters

ATK\$DSP_GET_COUNTERS obtains a list of internal counters maintained by ADSP. These counters describe the internal significant events which ADSP has incurred.

Format ATK\$DSP_GET_COUNTERS Refnum, CountList

Arguments **Refnum**

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which to obtain the counters. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

CountList

type: longword
access: write only
mechanism: by reference

The list of counters that will be returned in the form of a tagged list. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK\$K_CntLen bytes in length. The first byte of each entry is the tag, identifying the ADSP internal counter the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP internal counter follows the length field in the entry. If the CountList argument is omitted, an error is returned.

Conditions SS\$ _NORMAL Normal successful completion.

 ATK _INSNUMARG Insufficient number of arguments supplied.

 ATK _BADREFNUM Bad connection reference number.

ATK\$DSP_CLEAR_COUNTERS – Clear the ADSP internal counters

ATK\$DSP_CLEAR_COUNTERS clears ADSP's internal counters. These counters describe the internal significant events that ADSP has incurred.

Format ATK\$DSP_CLEAR_COUNTERS Refnum

Arguments **Refnum**

type: longword
access: read only
mechanism: by value

The reference number of the connection end whose counters will be cleared. Refnum must refer to a connection end previously established with ATK\$DSP_OPEN, ATK\$DSP_LISTEN, ATK\$DSP_ACCEPT, or ATK\$DSP_ESTABLISH. If Refnum is not specified, or is specified as zero, an error is returned.

Conditions

SS\$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.



Appendix A

Tuning Parameter Identifiers

The ADSP for VMS tuning parameter identifiers discussed in this appendix are

ATK\$K_ATTNBCKOFF
ATK\$K_ATTNINTRVL
ATK\$K_FWDRSTBCKOFF
ATK\$K_FWDRSTINTRVL
ATK\$K_OPENINTRVL
ATK\$K_OPENMAXRETRY
ATK\$K_RCVQCPYTHRESH
ATK\$K_RCVQMAXSIZE
ATK\$K_RESTARTBCKOFF
ATK\$K_RESTARTINTRVL
ATK\$K_RETRANSBCKOFF
ATK\$K_RETRANSINTRVL
ATK\$K_RETRANSTHRESH
ATK\$K_SNDQBUFSIZE
ATK\$K_SNDWDCLOBCKOFF
ATK\$K_SNDWDCLOINTRVL
ATK\$K_SNDWDCLOTHRESH
ATK\$K_TRANSTHRESH
ATK\$K_TRANSTIMERINTRVL

ATK\$K_ATTNBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The attention-message backoff rate defines how quickly ADSP backs off from retransmitting an unacknowledged attention message. The first retransmission occurs when the attention-message interval expires. The time interval between succeeding retransmissions is governed by the attention-message backoff rate. Each succeeding time interval is multiplied by the attention-message backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between successive retransmissions does not change.

ATK\$K_ATTNINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The attention-message retransmit interval defines the time interval between retransmissions of an ADSP attention message. When the client issues a send attention message request, ADSP builds an attention message, transmits it, and starts the attention-message retransmit timer. At each expiration of this timer, the attention message is retransmitted.

ATK\$K_FWDRSTBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The forward-reset backoff rate defines how quickly ADSP backs off from retransmitting an unacknowledged forward reset. The first retransmission of the forward reset occurs when the forward-reset interval expires. The time interval between succeeding retransmissions is governed by the forward-reset backoff rate. Each succeeding time interval is multiplied by the forward-reset backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between successive retransmissions does not change.

ATK\$K_FWDRSTINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The forward-reset retransmit interval defines how often ADSP retransmits a forward-reset message. When the client issues a forward-reset request, ADSP builds a forward-reset message and transmits it. The forward-reset retransmit timer is also started. At each expiration of this timer, the forward-reset message is retransmitted.

ATK\$K_OPENINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The retransmit open request interval defines the time interval between retransmissions of an open connection request. Open connection requests are sent in response to the reception of an open connection request from the remote end, or in response to an active open request from the local client (ATK\$DSP_OPEN or ATK\$DSP_ACCEPT). At each expiration of the retransmit open request interval, an open connection request message is transmitted to the remote end.

ATK\$K_OPENMAXRETRY

Field Length:	4 bytes	Min:	0
Usage:	unsigned longword	Max:	256
Default:	32		

This parameter defines the number of attempts ADSP makes to establish a connection with a remote end. Each time the retransmit open request interval expires, the open-retry count is incremented. Once this count reaches the value of the maximum retry parameter, the attempt to open the connection is aborted. Using a value of zero for this parameter forces ADSP to continue retransmitting open connection requests until the connection is actually established.

ATK\$K_RCVQCPYTHRESH

Field Length:	4 bytes	Min:	20 bytes
Usage:	unsigned longword	Max:	572 bytes
Units:	bytes	Default:	20 bytes

The receive-queue copy threshold determines whether ADSP tries to copy data received from the remote end into the last buffer in the receive queue (to avoid wasted space), or simply links the data into the receive queue. Data received from the remote end is in the form of a DDP datagram. If the amount of data in a received datagram is less than the receive-queue copy threshold, the data is copied into the last buffer in the receive queue; otherwise, the new datagram is linked. If there is no room in the previous buffer, or none exists, the datagram is linked regardless of the value of the receive-queue copy threshold.

ATK\$K_RCVQMAXSIZE

Field Length:	4 bytes	Min:	2048 bytes
Usage:	unsigned longword	Max:	65536 bytes
Units:	bytes	Default:	4096 bytes

The receive-queue maximum size defines the original size of the receive window. During connection establishment, the remote end is notified that the local end's receive queue is ATK\$K_RCVQMAXSIZE bytes in length.

ATK\$K_RESTARTBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The restart-sender backoff rate defines how quickly ADSP backs off from transmitting acknowledgment requests in order to restart the remote end's data transmission. The first ACK request is transmitted after the restart-sender interval expires. The time interval between succeeding ACK requests is governed by the restart-sender backoff rate. Each succeeding time interval is multiplied by the restart-sender backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK\$K_RESTARTINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The restart-sender interval defines how often ADSP transmits ACK request messages in order to notify the remote end that the receive window is open. When the receive window reopens after becoming closed, an ACK request message is transmitted and the restart-sender timer is started. At each expiration of the restart-sender interval, another ACK request message is transmitted. Once an acknowledgment is received from the remote end, the timer is stopped.

ATK\$K_RETRANSBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The retransmit backoff rate defines how quickly ADSP backs off from transmitting ACK requests in order to retransmit data. The first ACK request is transmitted after the retransmit interval expires. The time interval between succeeding ACK requests is governed by the retransmit backoff rate. Each succeeding time interval is multiplied by the retransmit backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK\$K_RETRANSINTRVL

Field Length:	4 bytes	Min:	0 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The retransmit interval defines how long ADSP waits for the acknowledgment of transmitted data. Once a datagram is transmitted to the remote end, the retransmit interval timer is started. If an acknowledgment is not received before the interval expires, an ACK request message is transmitted to the remote end, and the timer is restarted. Any acknowledgment received after the original expiration of the timer causes all unacknowledged data to be retransmitted. The unacknowledged data is assumed to be lost. The timer is then restarted at its original interval. The timer is stopped when all outstanding data bytes are acknowledged by the remote end.

ATK\$K_RETRANSTHRESH

Field Length:	4 bytes	Min:	0
Usage:	unsigned longword	Max:	10
Units:	datagrams	Default:	4

The retransmit advice threshold defines when the local ADSP implementation sends a retransmit advice message to the remote end. If the count of consecutive out-of-sequence datagrams received by the local end reaches the value of the retransmit advice threshold, a retransmit advice message is transmitted. The reception of an in sequence datagram resets the counter.

ATK\$K_SNDQBUFSIZE

Field Length:	4 bytes	Min:	100 bytes
Usage:	unsigned longword	Max:	16384 bytes
Units:	bytes	Default:	112 bytes

The send queue buffer size defines the size of buffers ADSP allocates for the send queue in response to ATKSDSP_PUT_BYTE requests. Each Put Byte request issued to ADSP attempts to copy the data byte into the last buffer in the send queue. When this buffer is full, a new buffer is allocated, which is ATK\$K_SNDQBUFSIZE bytes in size.

ATK\$K_SNDWDWCLOBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The send window closed backoff rate defines how quickly ADSP backs off from transmitting ACK requests in order to open the send window. The first ACK request is transmitted after the send window closed interval expires. The time interval between succeeding ACK requests is governed by the send window closed backoff rate. Each succeeding time interval is multiplied by the send window closed backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK\$K_SNDWDWCLOINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The send window closed interval defines how often ADSP solicits the remote end for acknowledgment information once the send window closes. Once the send window closes, the send window closed timer is started. At each expiration of the send window closed interval, an ACK request message is transmitted to the remote end. Once an acknowledgment that opens the send window is received, the timer is stopped.

ATK\$K_SNDWDWCLOTHRESH

Field Length:	4 bytes	Min:	0 bytes
Usage:	unsigned longword	Max:	8192 bytes
Units:	bytes	Default:	1024 bytes

The send window closing threshold defines the point in the send queue where ADSP will send an ACK request in anticipation of the send window closing. When transmitting data to the remote end, ADSP passes through the send queue until it reaches the point where the send window closes and it cannot send anymore data. At this point, ADSP must wait for the reception of an acknowledgment from the remote end, which opens the send window. The send window closing threshold marks the point in the send queue preceding the send window closed point. When ADSP reaches this point, it sets the ACK request bit in the ADSP header of the next transmitted datagram. This request solicits an acknowledgment from the remote end, which hopefully advances the send window before the local end reaches the send window closed point. An ACK request is always sent immediately when the send window closes.

ATK\$K_TRANSTHRESH

Field Length:	4 bytes	Min:	0 bytes
Usage:	unsigned longword	Max:	572 bytes
Units:	bytes	Default:	112 bytes

The transmit threshold is the number of unsent data bytes ADSP accumulates in the send queue before transmitting the data to the remote end. A counter is incremented each time a byte is inserted into the send queue. When this counter reaches the value of the transmit threshold, a datagram is built and the data is transmitted. The counter is then reset to zero.

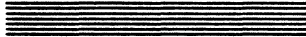
ATK\$K_TRANSTIMERINTRVL

Field Length:	4 bytes	Min:	0 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The transmit timer interval defines how long ADSP waits for the transmit threshold to be reached. The transmit timer is started when the first data byte is inserted into the send queue. If the transmit timer interval expires before the transmit threshold is reached, all unsent data bytes are transmitted to the remote end. The transmit threshold counter is then reset to zero. If the transmit threshold is reached before the interval expires, the timer is either restarted or stopped. If there remains any unsent data in the send queue, the timer is restarted; otherwise it is stopped.



Appendix B

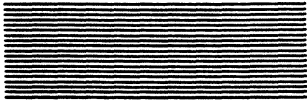


Connection Status Identifiers

Each of the connection status identifiers described below is used as an unsigned longword with a field length of 4 bytes.

Identifier	Description
ATK\$K_ATTNMSGWAIT	A logical value indicating whether an attention message is waiting in the receive queue. If set to nonzero, an attention message is waiting. If set to zero, the receive queue is empty.
ATK\$K_ATTNRCVSEQ	The sequence number of the next attention message that will be accepted from the remote end.
ATK\$K_ATTNSNDSEQ	The sequence number of the next attention message to be transmitted to the remote end.
ATK\$K_FIRSTRTMTSEQ	The sequence number of the first unacknowledged byte in the send queue.
ATK\$K_RCVSEQ	The sequence number of the next byte that will be accepted from the remote end.
ATK\$K_RCVWDW	The current size of the local receive window.
ATK\$K_SNDSEQ	The 32-bit sequence number of the first untransmitted byte in the send queue.

ATK\$K_SNDWDWSEQ	The sequence number of the last byte in the send queue that may be transmitted to the remote end (that is, the send window close point).
ATK\$K_STATE	The state of the connection. This field can take on one of the values that follow.
ATK\$K_INVALID	The connection control block is invalid.
ATK\$K_LISTEN	The connection end is passively waiting for an incoming open connection request.
ATK\$K_OPENING	The connection end is actively seeking to open a connection with a remote end.
ATK\$K_MYHALFOPEN	The connection end has obtained the state information of the remote end, and is actively seeking to open the connection.
ATK\$K_HISHALFOPEN	The remote connection end has obtained the local end's state information and the local end is actively seeking to open the connection.
ATK\$K_OPEN	The connection is operational.
ATK\$K_TORNDOWN	The connection is half-open since the remote end is unreachable.
ATK\$K_CLOSING	The local client has issued a close request, and the connection is waiting for the send queues to empty.
ATK\$K_CLOSED	A close advice has been received from the remote end, and the connection end is waiting for the receive queues to empty.



Appendix C



ADSP Internal Counter Identifiers

Each of the internal counter identifiers described below is used as an unsigned longword with a field length of 4 bytes.

Identifier	Description
ATK\$K_ATTNACKRCVD	The number of attention message acknowledgments received from the remote end.
ATK\$K_ATTNACKACPTD	The number of attention message acknowledgments accepted by the local end.
ATK\$K_PROBERCVD	The number of probe messages received from the remote end.
ATK\$K_ACKRCVD	The number of explicit ACK messages received from the remote end.
ATK\$K_FWDRSTRCVD	The number of forward-reset messages received from the remote end.
ATK\$K_FWDRSTACPTD	The number of forward-reset messages accepted by the local end.
ATK\$K_FWDRSTACKRCVD	The number of forward-reset acknowledgment messages received from the remote end.

ATK\$K_FWDRSTACKACPTD	The number of forward-reset acknowledgments accepted by the local end.
ATK\$K_ATTNRCVD	The number of attention messages received from the remote end.
ATK\$K_ATTNACPTD	The number of attention messages accepted by the local end.
ATK\$K_DATAARCVD	The number of data messages received from the remote end.
ATK\$K_DATAACPTD	The number of data messages accepted by the local end.
ATK\$K_ACKFIELDCHKD	The number of times the acknowledgment fields of the ADSP header were examined.
ATK\$K_ACKNRSFIELDACPTD	The number of times the Next Receive Sequence Number field was accepted by the local end.
ATK\$K_ACKSWSFIELDACPTD	The number of times the calculated Send Window Sequence Number field was accepted by the local end.
ATK\$K_ACKREQSTD	The number of times an acknowledgment was requested by the remote end.
ATK\$K_LOWMEM	The number of times the local end encountered low memory conditions.
ATK\$K_OPNREQEXP	The number of times the open request timer expired.
ATK\$K_PROBEEXP	The number of times the probe timer expired.
ATK\$K_FWDRSTEXP	The number of times the forward-reset retransmission timer expired.
ATK\$K_ATTNEXP	The number of times the attention-message retransmission timer expired.
ATK\$K_TRANSEXP	The number of times the transmit interval timer expired.
ATK\$K_RETRANSEXP	The number of times the retransmit unacknowledged data timer expired.
ATK\$K_SNDWDWCLOEXP	The number of times the send window closed timer expired.
ATK\$K_RESTARTEXP	The number of times the restart-sender timer expired.

ATK\$K_RESLOWEXP

The number of times the resources-low timer expired.

ATK\$K_RETRANSRCVD

The number of times a retransmit advice message was received.

